Improving KantanMT Training Efficiency with `fast_align`

Dimitar Shterionov dimitars@kantanmt.com
KantanLabs, Dublin, Ireland
Jinhua Du jinhua.du@adaptcentre.ie
ADAPT Centre, DCU, Dublin, Ireland
Marc Anthony Palminteri marcp@kantanmt.com
KantanMT.com, Dublin, Ireland
Laura Casanellas laurac@kantanmt.com
KantanMT.com, Dublin, Ireland
Tony O'Dowd tonyod@kantanmt.com
KantanMT.com, Dublin, Ireland
Andy Way andy.way@adaptcentre.ie
ADAPT Centre, DCU, Dublin, Ireland

Abstract

In recent years, statistical machine translation (SMT) has been widely deployed in translators' workflow with significant improvement of productivity. However, prior to invoking an SMT system to translate an unknown text, an SMT engine needs to be built. As such, building speed of the engine is essential for the translation workflow, i.e., the sooner an engine is built, the sooner it will be exploited.

With the increase of the computational capabilities of recent technology the building time for an SMT engine has decreased substantially. For example, cloud-based SMT providers, such as KantanMT, can built high-quality, ready-to-use, custom SMT engines in less than a couple of days. To speed-up furthermore this process we look into optimizing the word alignment process that takes place during building the SMT engine. Namely, we substitute the word alignment tool used by KantanMT pipeline – `Giza++` – with a more efficient one, i.e., `fast_align`.

In this work we present the design and the implementation of the KantanMT pipeline that uses `fast_align` in place of `Giza++`. We also conduct a comparison between the two word alignment tools with industry data and report on our findings. Up to our knowledge, such extensive empirical evaluation of the two tools has not been done before.

1 Introduction

In recent years, statistical machine translation (SMT) systems have been widely deployed in translators' workflows with significant improvements in productivity. KantanMT is a cloud-based SMT platform that allows its clients to train SMT engines that are customized for their specific translation tasks by using their own data. Many factors contribute to the quality of service provided by KantanMT, e.g. the speed for training an SMT engine with KantanMT is an essential factor as it determines how fast clients can commence translating.[1]

[1] To date, SMT engines built using 250 million words of training data can be built with KantanMT in about 3 days.

The KantanMT platform employs a cloud-based architecture that has two main components: (i) an interface component to process, coordinate and distribute job requests and (ii) a collection of processing steps to execute build, translation or analysis jobs. The architecture of KantanMT is based on the Amazon Web Services (AWS)[2]. For any job request the interface allocates a machine from the cloud. The machine is set-up to comply with KantanMT requirements and used afterwards to execute the specific job. For a build job, the KantanMT training pipeline is composed of 14 steps.

Crucial for the efficiency of the KantanMT training pipeline is word alignment. Word alignment is the task of identifying word-level translation relations between a source text and its translation. Naturally, to date the KantanMT pipeline has been using `Giza++` (Och and Ney (2003)) – the most common word-alignment tool used by the SMT community – for word alignment. An alternative to `Giza++` is `fast_align` (Dyer et al. (2013)), a simple, fast, yet effective tool to perform word alignment. Dyer et al. (2013) show that `fast_align` is about 10 times faster than IBM Model 4 (Brown et al. (1993)). Moreover, `fast_align` leads to translation performance comparable to MT engines trained using `Giza++` (Dyer et al. (2013)). Accordingly, with the aim of reducing the training time of KantanMT engines, we introduced `fast_align` into the KantanMT pipeline in place of `Giza++`. Improved training times would lead to better quality of service as well as reduced resource allocation, an important issue for any cloud-based system.

In this work we present the integration of `fast_align` into the KantanMT training pipeline. We focus on (i) our collaborative approach to integrating the `fast_align` tool into a live production system; (ii) the improvements in training time; and (iii) a comparison of the translation quality between MT engines built with `Giza++` and with `fast_align`.[3]

2 Training KantanMT engines with `Giza++`

2.1 KantanMT training pipeline

Once a building request has been received and a dedicated machine has been allocated, there are 14 processing steps that take place in order to train a KantanMT engine. Each processing step applies on the output from the preceding step and provides input for the next[4]. These steps can be divided into 5 major stages:

1. **Instance setup.** Required software is downloaded and installed; bilingual and monolingual data is retrieved and verified.

2. **Data preprocessing.** Once the data is downloaded and verified it is subjected to preprocessing, cleansing and partitioning. The bilingual data is divided into three sets – a training, a tuning and a test set – that are used for training and optimizing the engine.

3. **Building.** Three models are built during this stage: (i) a language model that captures the linguistic aspects of the target language and aims at improving MT output; (ii) a recaser model to set the correct letter casing in the MT output and (iii) a translation model used for decoding unseen text. Monolingual data (in the target language) is often used to improve the quality of the language model. The KantanMT platform employs the open-source toolkit Moses (Koehn et al. (2007)) to train the language, the translation and the recaser models.

[2] `https://aws.amazon.com/`

[3] To the best of our knowledge, such an extensive empirical evaluation of these two word-alignment approaches with industry data has not been performed to date.

[4] That is why we refer to the KantanMT architecture as a *pipeline* architecture.

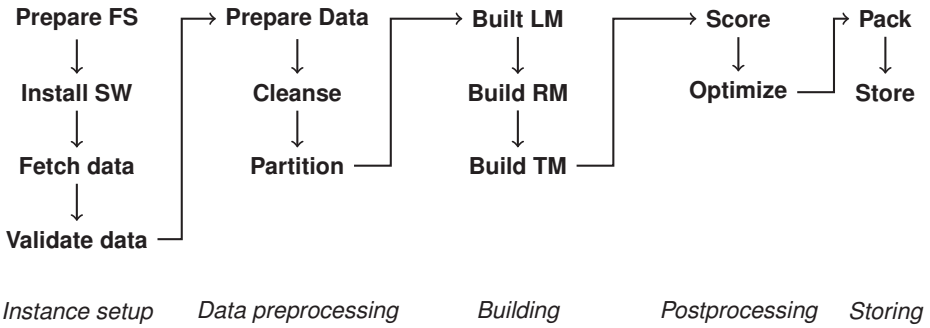

Figure 1: KantanMT training pipeline.

4. **Engine postprocessing.** After the engine is built it is scored by calculating three evaluation metrics: (i) BLEU score (Papineni et al. (2002)); (ii) F-Measure (van Rijsbergen (1979); Melamed (1995)) and (iii) Translation Error Rate (TER) (Snover et al. (2006)). If required, the engine is also optimized by using the tuning subset from the training data.

5. **Storing.** The models, configuration files and scores are packed and stored for future use.

Word alignment is invoked during building the translation model. To compute the word alignment is one of the computationally most expensive tasks in the building step. Up to date, KantanMT was using `Giza++` to perform word alignment during this step.

Figure 1 shows the original (i.e., using the `Giza++` word-alignment tool) KantanMT pipeline for training an engine.

2.2 Faster word-alignment for faster end-user delivery

`Giza++` implements IBM models 1 to 5 (Brown et al. (1993)) as well as an HMM word alignment model (Och and Ney (2003)). IBM models 1 and 2 are computationally inexpensive, however, higher IBM models increase the complexity of training the models by adding additional components. For example, IBM Model 3 introduces a fertility model which can address the issue of an input word producing multiple target words or zero words[5].

A software platform oriented to provide MT services to industry, such as KantanMT, needs to meet its clients requirements for quick service delivery. Moreover, there is a twofold gain in improving engine build time: (i) faster delivery to the end-user – the faster MT engines are trained, the sooner the client may start exploiting them; and (ii) optimized resource allocation – allocated hardware is released sooner and can be ready to use for new tasks faster.

A recent project, conducted by the ADAPT centre[6] (Du et al. (2015)), implements an SMT pipeline where `Giza++` is substituted by *fast_align* (Dyer et al. (2013)). Motivated by the results from their experiments and aiming to achieve faster delivery to the end-user as well as improved resource allocation, we integrate the *fast_align* word alignment tool in the KantanMT platform.

3 KantanMT pipeline with *fast_align*

In order to integrate *fast_align* into the KantanMT training pipeline, an absolute prerequisite is that it needs to be 100% compatible with the already used MT pipeline. This has two require-

[5]For more information about the IBM models and their complexity as well as about HMM models we refer the interested reador to (Koehn (2010)).

[6]`http://adaptcentre.ie`

ments: (i) *no* task that KantanMT performs should see *any* degradation in performance; and (ii) the user experience should *not* be negatively impacted.

3.1 A collaborative approach for industry-standard software development

In order to ensure the aforementioned requirements, we devised a systematic development approach that teamed state-of-the-art academic knowledge and experience with industry-leading software development and quality assessment (QA). We developed our solution following an AGILE-based methodology[7] that involved three main stages:

1. **Design**. During the design stage we first analysed the current KantanMT pipeline and identified the software requirements towards the pipeline for integrating a new word alignment tool. We then identified the software requirements of *fast_align*. Joining the industry experience and the academic know-how we formulated the system design, i.e., the functional and technical specifications of the modified KantanMT pipeline[8]. Furthermore, we reviewed and accepted any licences of additionally required software.

2. **Implementation**. Upon approval of the system design we implemented the new pipeline according to the steps defined in the technical specification document. As a first part of the quality assessment (QA) strategy we performed a series of tests to verify that each component of the pipeline, including the newly introduced ones, works as designed (i.e., alpha testing).

3. **Quality Assessment**. Our QA strategy involved alpha testing performed in parallel with the initial implementation; beta testing and life testing. While the alpha testing aimed to verify the coherence of the modified pipeline, the latter two parts aimed to ensure that the integrity of the whole system is intact. In addition, during the life testing we focused on the efficiency and user experience of the platform, i.e., empirical evaluation.

The duration of the project for integrating `fast_align` into the KantanMT training pipeline is 4 weeks plus 2 additional weeks for empirical evaluation.

3.2 System requirements

In order to incorporate `fast_align` into the KantanMT pipeline we first ensure that all software requirements of `fast_align` are met (see `http://www.cdec-decoder.org/guide/` for details). Next we need to ensure that the input data requirements are met as well. In the original KantanMT pipeline (i.e., using `Giza++`) we use two files that represent the parallel corpus: one for the source part of the corpus and another for the target part[9]. The *fast_align* tool uses a different input format. It requires a single file in which each line contains both the source and target parts for one sentence, separated by a triple pipe (`|||`)[10]. Example 3.1 shows such formatting for German source sentences and its English translation[11].

Example 3.1 *doch jetzt ist der Held gefallen* . `|||` *but now the hero has fallen* .

We use the `paste_files.pl` script from the `cdec`[12] tool collection to join the two files in one and therefore ensure the correct formatting for `fast_align`. We invoke `paste_files.pl` right after the data cleansing (see Figure 1).

[7] `http://agilemethodology.org/`

[8] The functional specifications of the pipeline were outlined in a functional specification document; the technical specifications – in a technical specification document.

[9] During the data preparation step (see Figure 1) the corpus is encoded in UTF-8.

[10] `http://www.cdec-decoder.org/guide/fast_align.html`

[11] `http://www.cdec-decoder.org/guide/fast_align.html`

[12] `http://www.cdec-decoder.org/`

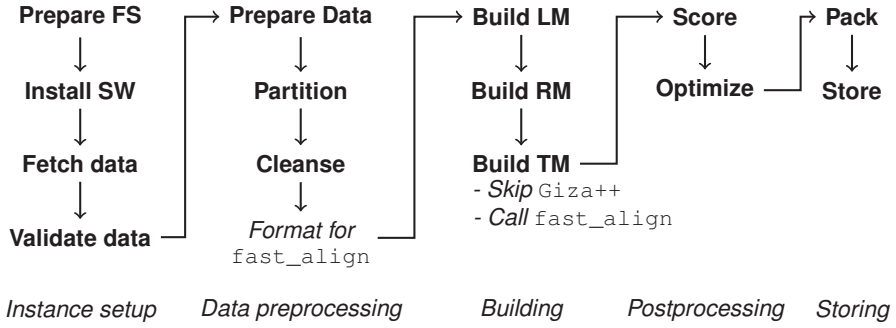

Figure 2: KantanMT training pipeline modified to invoke `fast_align`.

We run the `fast_align` tool in forward (source–target) and reversed (target–source) directions. Each direction generates asymmetric alignments, i.e., by treating either the source or target language in the parallel corpus as primary language being modeled. These two directional `fast_align` will generate slightly different alignments and need to be then symmetrized.We use the `atools` script from the `cdec` collection. The output from `atools` produces a word alignment in the widely-used $i-j$ Pharaoh format (Koehn (2003)). In this format, the pair $i-j$ indicates that the i^{th} word (zero-indexed) of the source language is aligned to the j^{th} word of the target language. Example 3.2 shows the word alignment in Pharaoh format for the sentences in Example 3.1.

Example 3.2 *0-0 1-1 2-4 3-2 4-3 5-5 6-6*

The word alignment in the Pharaoh format is compatible with the steps in the building process that follow the word alignment step.

3.3 System architecture

In order to integrate `fast_align` we modify the original KantanMT pipeline as follows:

1. Incorporate additional data processing step where we invoke `paste_file.pl` to join the two files of the parallel corpus in one according to the requirements of `fast_align` (see Section 3.2).
2. During the build of the translation model we skip `Giza++` and invoke `fast_align`.
3. Invoke `atools` to symmetrise the output of `fast_align`.

The rest of the pipeline remains the same. Our modifications are outlined in Figure 2.

The modularity of our pipelines – the original KantanMT training pipeline and the modified one – allow an easy switching between the two, i.e., between training an engine with `fast_align` and with `Giza++`. To do so we use a global variable that acts as a switch between `fast_align` and `Giza++`. This ensures (i) higher control on the pipeline; (ii) quick and easy rollback in case of unforeseen system issues.

4 Empirical evaluation

In order to determine the effectiveness of the `fast_align` word alignment tool, a series of experiments were conducted in a closed, controlled system separate from the normal workflow of KantanMT. The goal of running these experiments was to quantify the performance of the KantanMT training pipeline as well as the translation quality of engines trained using `fast_align` as compared to the original pipeline that uses `Giza++`.

Language pair	*Relative size*	*Word count*	*Unique word count*	*Engine reference*
English-French	Small	781 075	42 563	EN-FR-small
English-French	Large	109 379 800	1 008 696	EN-FR-large
English-German	Small	786 981	42 648	EN-DE-small
English-German	Large	138 119 563	1 084 485	EN-DE-large
English-Spanish	Small	861 557	44 375	EN-ES-small
English-Spanish	Large	154 169 102	1 119 475	EN-ES-large
English-Italian	Small	924 331	38 506	EN-IT-small
English-Italian	Large	104 196 079	914 889	EN-IT-large
English-Chinese	Small	810 134	33 281	EN-ZH-small
English-Chinese	Large	58 274 131	550 862	EN-ZH-large

Table 1: Data sets used in our comparison experiments.

4.1 Set-up

Our tests involved building 10 engines from 10 different data sets with both `Giza++` and `fast_align`. These data sets are of varying sizes and language pairs. The data is from legal and financial domain; it is part of the KantanLibrary^{TM}[13]. At the end of the experiments, speed, performance, accuracy, and stability of each engine were compared to their counterpart for a direct `Giza++`-to-`fast_align` assessment. Details about the used data sets are presented in Table 1.

We decided to use small data sets in order to test whether the alignment tool would perform better when given a small data set as compared to larger data sets. The engines built with the larger data sets were used to derive close to realistic estimates on the overall engine performance – i.e., automatic evaluation metrics and building time – when comparing the two alignment tools. Typically, a specialised engine is built on around 10 million words. The language pairs were selected based on the availability of professional linguists, translators or native speakers who can evaluate the quality of translated files.

4.2 Experiments

4.2.1 Experiment 1 – Time consumption

The objective of *Experiment 1* is to judge the speed of the KantanMT training pipeline. First we broke down the steps to building an engine from the moment the training data is prepared to when the final package is completed (see Figure 1 and Figure 2). Each step was launched manually, monitored, and timed for all 10 engines for both pipelines – with `Giza++` and with `fast_align`. We then sum the time of each individual step in order to compute the total training time. Table 2 summarises our results from timing each individual step. It shows the time gain (T^{+}) for building an engine with `fast_align` (T_{fa}) as compared to an engine built using `Giza++` (T_{ga}): $T^{+} = \frac{T_{ga} - T_{fa}}{T_g a}$.

Table 2 reveals that each engine experienced a decrease in building time when using `fast_align`, with the time gain being between 48% and 73%. We ought to noted that these times do not account for the initial training data upload time, the instance setup, the data preparation, or the job clean-up which, when dealing with large engines, can add a significant amount of time to a build, i.e., can take around $40 - 60\%$ of the total job time. In order to estimate the impact of the new word alignment tool under life conditions we also measured the time for training an engine launched from the online interface of the platform. Table 2 summerizes our results from timing the engine build including the data upload time, instance setup, etc.

[13] KantanLibrary^{TM} is the collection of industry-standard data that KantanMT provides to their clients.

	Specific tasks			Complete pipeline		
Engine	*Giza++ (hh:mm:ss)*	*fast_aling (hh:mm:ss)*	*Time gain*	*Giza++ (hh:mm:ss)*	*fast_aling (hh:mm:ss)*	*Time gain*
EN-FR-small	00 : 09 : 23	00 : 03 : 49	59.00%	00 : 25 : 00	00 : 18 : 00	28.00%
EN-FR-large	10 : 35 : 11	04 : 02 : 14	62.00%	24 : 49 : 00	09 : 04 : 00	63.00%
EN-DE-small	00 : 10 : 06	00 : 03 : 57	61.00%	00 : 28 : 00	00 : 17 : 00	39.00%
EN-DE-large	15 : 33 : 43	04 : 13 : 57	73.00%	26 : 44 : 00	10 : 28 : 00	61.00%
EN-ES-small	00 : 10 : 21	00 : 04 : 20	58.00%	00 : 27 : 00	00 : 20 : 00	26.00%
EN-ES-large	14 : 07 : 21	04 : 54 : 12	65.00%	26 : 04 : 00	07 : 58 : 00	69.00%
EN-IT-small	00 : 11 : 03	00 : 04 : 32	59.00%	00 : 29 : 00	00 : 22 : 00	24.00%
EN-IT-large	11 : 09 : 32	05 : 46 : 41	48.00%	19 : 27 : 00	06 : 47 : 00	65.00%
EN-ZH-small	00 : 10 : 07	00 : 04 : 35	55.00%	00 : 20 : 00	00 : 16 : 00	20.00%
EN-ZH-large	10 : 08 : 16	03 : 34 : 13	65.00%	13 : 18 : 00	06 : 55 : 00	48.00%
		Average:	60.50%		Average:	44.30%

Table 2: Summary of the results from *Experiment 1*: time comparison.

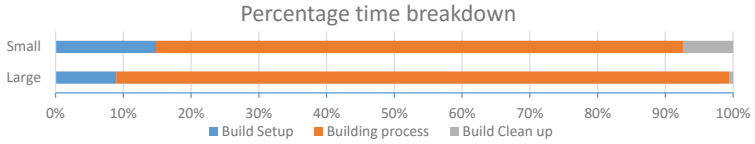

Figure 3: Percentage time taken to complete each building phase.

These results show a more accurate interpretation as to how long a build job would take for a user. For the smaller engines, the time gain from using `fast_align` has a smaller impact due to the fact that the setup and shutdown times do not change, taking a larger proportion of the total job time. In Figure 3 we show an time for initialization, training and cleanup for a random small and a large engines.

Table 2 and Figure 3 indicate that the higher the word count of the training data the more influenced the training is by the specific training steps of the pipeline, and in particular by the word alignment. That is why, for larger engines `fast_align` typically leads to higher time gain.

4.2.2 Experiment 2 – Automatic quality estimation

One of the final step in KantanMT training pipeline is to score the engines (see Figure 1). We compare the *F-Measure*, *BLEU* and *TER* scores computed at that step for each engine in order to determine the relative quality between engines built with `Giza++` and engines built with `fast_aling`. We summarize our results in Table 3.

For engines that were built with `fast_align` we notice maximum score decrease of 4.5 points (in the EN-DE-large engine) and maximum score increase of 2.2 points (in the EN-ES-large engine). The average difference is 1 point in favour of engines built with `Giza++`.

Regarding, BLEU score, we notice a maximum decrease of 2.8 points (in the EN-DE-small engine) and a maximum increase of 3.6 points (in the EN-DE-large engine). The average

	F-Measure				*BLEU*				*TER*			
Engine	`Giza++`	`fast_align`	*Absolute*	*Relative*	`Giza++`	`fast_align`	*Absolute*	*Relative*	`Giza++`	`fast_align`	*Absolute*	*Relative*
EN-FR-small	63.0	61.8	−1.2	1.90%	62.7	60	−2.7	4.31%	51.9	53.5	1.6	−3.08%
EN-FR-large	70.4	69.5	−0.9	1.28%	61.8	62.2	0.4	−0.65%	42.7	43.5	0.8	−1.87%
EN-DE-small	57.3	58.6	1.3	−2.27%	55.6	59.2	3.6	−6.47%	58.9	55.1	−3.8	6.45%
EN-DE-large	70.8	66.3	−4.5	6.36%	66.2	63.4	−2.8	4.23%	43.7	49.6	5.9	−13.50%
EN-ES-small	69.5	67.1	−2.4	3.45%	59.2	56.9	−2.3	3.89%	44.9	48.6	3.7	−8.24%
EN-ES-large	73.7	75.9	2.2	−2.99%	60.5	63.5	3.0	−4.96%	40.2	37.2	−3	7.46%
EN-IT-small	61.9	61.0	−0.9	1.45%	54.2	53	−1.2	2.21%	52.6	54.4	1.8	−3.42%
EN-IT-large	71.0	66.6	−4.4	6.20%	60.5	61.3	0.8	−1.32%	41	44.6	3.6	−8.78%
EN-ZH-small	75.4	76.5	1.1	−1.46%	44.2	45.3	1.1	−2.49%	43.9	41.5	−2.4	5.47%
EN-ZH-large	74.7	74.4	−0.3	0.40%	53.7	52.2	−1.5	2.79%	48.7	48.8	0.1	−0.21%

Table 3: Summary of the results from *Experiment 2*: F-Measure, BLUE and TER scores.

difference is 0.16 points in favour of engines built with `Giza++`.

The last three columns in Table 3 refer to the TER score. It shows a maximum increase of 5.9 points (in the EN-DE-large engine) and maximum decrease of 3.8 points (in the EN-DE-small engine). On average there is an increased by 0.83 points for engines built with `fast_align`

Although we notice a general tendency of automatic quality measurements to decrease for engines built with `fast_align` (when compared to `Giza++`) we outh to point: (i) the fluctuation we observe in Table 3 indicates that there is no sensible change in quality and (ii) the fact that for different language pairs and data set sizes the scores alternate between `fast_align` and `Giza++` indicates that under specific conditions `fast_align` is better and under other conditions `Giza++` is better (the specifics of these conditions are out of the scope of this project and remain a topic for future research).

Experiment 3 In order to be get a more specific estimate of the quality of engines built with `fast_align` as compared to ones built with `Giza++` we involved professional translators and native speakers (members of the KantanMT Professional Services department as well as the ADAPT centre at DCU) for human evaluation. We performed a blind comparison of 4 documents translated by engines built with `fast_align` and with `Giza++` for each of the language pairs and data set sizes. The human evaluators would indicate which of the documents they considered to be better in terms of accuracy, fluency, and overall quality. The objective was to determine which engines translation would read easier when assessed by a person who is fluent in the target language.

For the small engines, all of translators independently said that the translation quality from engines built with `fast_align` is the same as from engines built with `Giza++`. In practice, due to the quantity of training data used by the smaller engines (being too small to translate to a high standard) the overall quality for all pairs was rather low. Therefore, we consider the results from the larger engines of practical value and focus on their analysis.

The translators had to answer two questions:

1. Which document has higher language quality?
2. Does the quality respond to their requirements as translators/native speakers?

All translators stated that both sets of documents were of a very high quality and that they cannot make a distinctive decision. With one exception: for the EN-FR-large engine, had translators disagreed as to which set of documents were better.

All translators consider both groups of translation documents to match their expectations. They also claimed that there was very little difference between documents, with mainly minor grammatical errors, e.g., two words in a long segment being in the wrong order. The results of the blind test were satisfactory to our requirements for the project.

5 Conclusions

In this work we substituted the word alignment tool used in the KantanMT pipeline – `Giza++` – with the more efficient and yet effective `fast_align`. The latter has already been successfully incorporated in other industrial products and has shown promising results. In our design and implementation strategy we combined industry established software development practices with academic know-how to effectively modify a large-scale online platform such as KantanMT with **no** downtime or decay in KantanMT services.

To assess the impact of the new word alignment method into the efficiency of the KantanMT pipeline as well as the quality of the built engines we performed a series of tests with industry based data. Our results show an average speed-up of 60% and comparable quality to engines built with `Giza++`.

Our experiments also show that for languages that differ substantially in the word order (such as English and German) `fast_align` may lead to a slight decrease in quality (according to automatic measures). In the future, we aim to carry out more investigation on word alignment results from `fast_align` and examine possible solutions of improving SMT performance.

References

Brown, P. F., Della-Pietra, S. A., Della-Pietra, V. J., and Mercer, R. L. (1993). The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, 19(2):263–311.

Du, J., Moorkens, J., Srivastava, A., Lauer, M., Way, A., Maldonado, A., and Lewis, D. (2015). D4.3: Translation project – level evaluation distribution: Public federated active linguistic data curation (falcon).

Dyer, C., Chahuneau, V., and Smith, N. A. (2013). A simple, fast, and effective reparameterization of IBM model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648, Atlanta, Georgia, USA.

Koehn, P. (2003). Pharaoh: a beam search decoder for phrase-based statistical machine translation models: User manual and description for version 1.2.

Koehn, P. (2010). *Statistical Machine Translation*. Cambridge University Press, New York, NY, USA, 1st edition.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., and Herbst, E. (2007). Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics: Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic.

Melamed, I. D. (1995). Automatic evaluation and uniform filter cascades for inducing n-best translation lexicons. In *Proceedings of the third Workshop on Very Large Corpora*, pages 184–198, Cambridge, Massachusetts, USA.

Och, F. J. and Ney, H. (2003). A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA.

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., and Makhoul, J. (2006). A study of translation edit rate with targeted human annotation. In *In Proceedings of Association for Machine Translation in the Americas*, pages 223–231, Cambridge, Massachusetts, USA.

van Rijsbergen, C. J. (1979). *Information Retrieval*. Butterworth.